AVL Trees

Theresa Migler-VonDollen

Running Times for Binary Search Trees

How long did it take to accomplish the following operations on binary search trees?

- *find:* $O(\text{height of tree}) = O(n)$.
  - This is because on average the height of a binary tree is $\log n$. But the worst case height is n.
- *insert:* $O(n)$
- *delete:* $O(n)$
- *isEmpty:* $O(1)$
- *findMin:* $O(n)$
- *findMax:* $O(n)$

What if we could make our binary search trees shorter?

AVL Trees

Definition

An *AVL* tree is a self-balancing binary search tree.

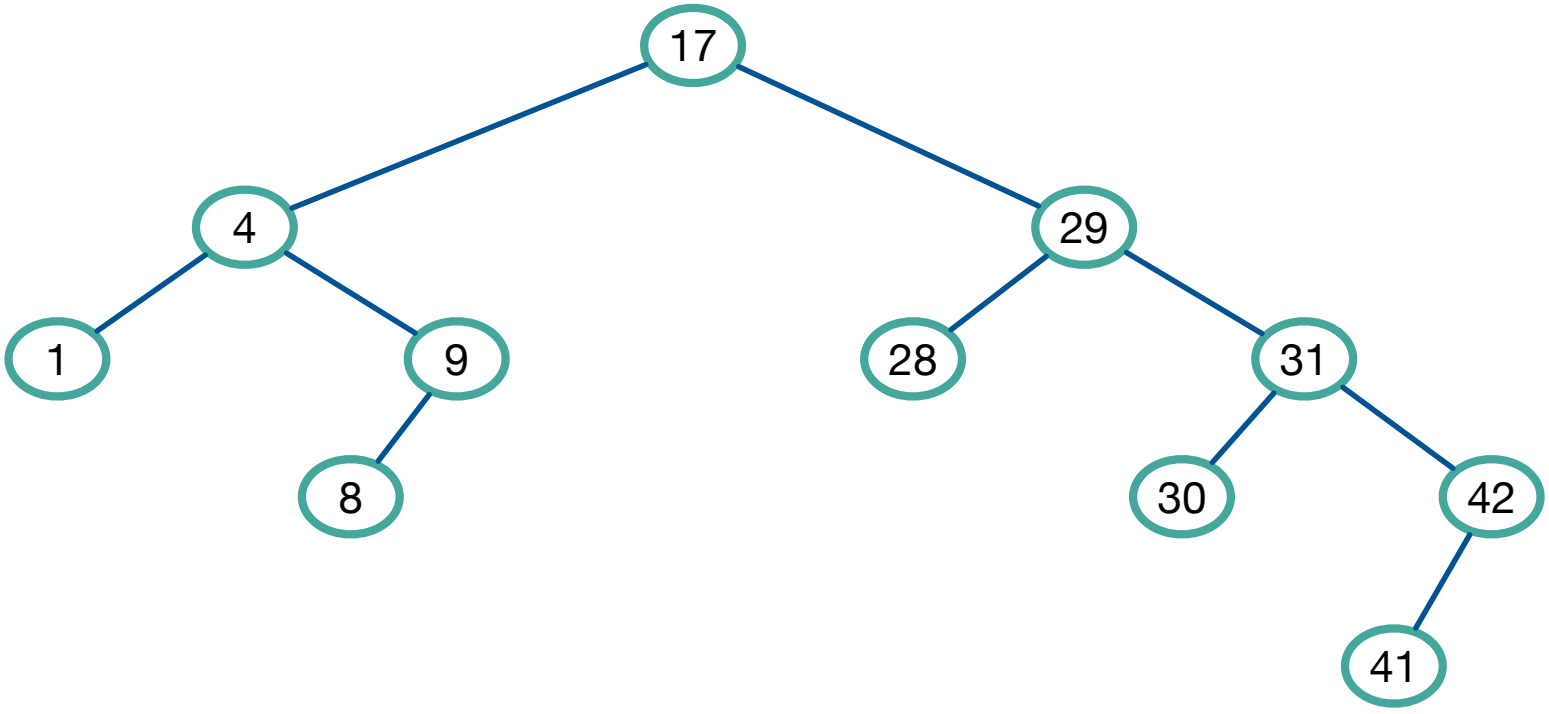
- This type of tree was created by Georgy Adelson-Velsky and Evgenii Landis in 1962.

Recall that the *height* of a vertex is the length of the longest downward path to a leaf from that vertex. The height of the tree is the height of the root.
In an AVL tree, the heights of the two child subtrees of any vertex differ by at most one.

- How long will the *find* operation take?
- *insert*?
- *delete*?

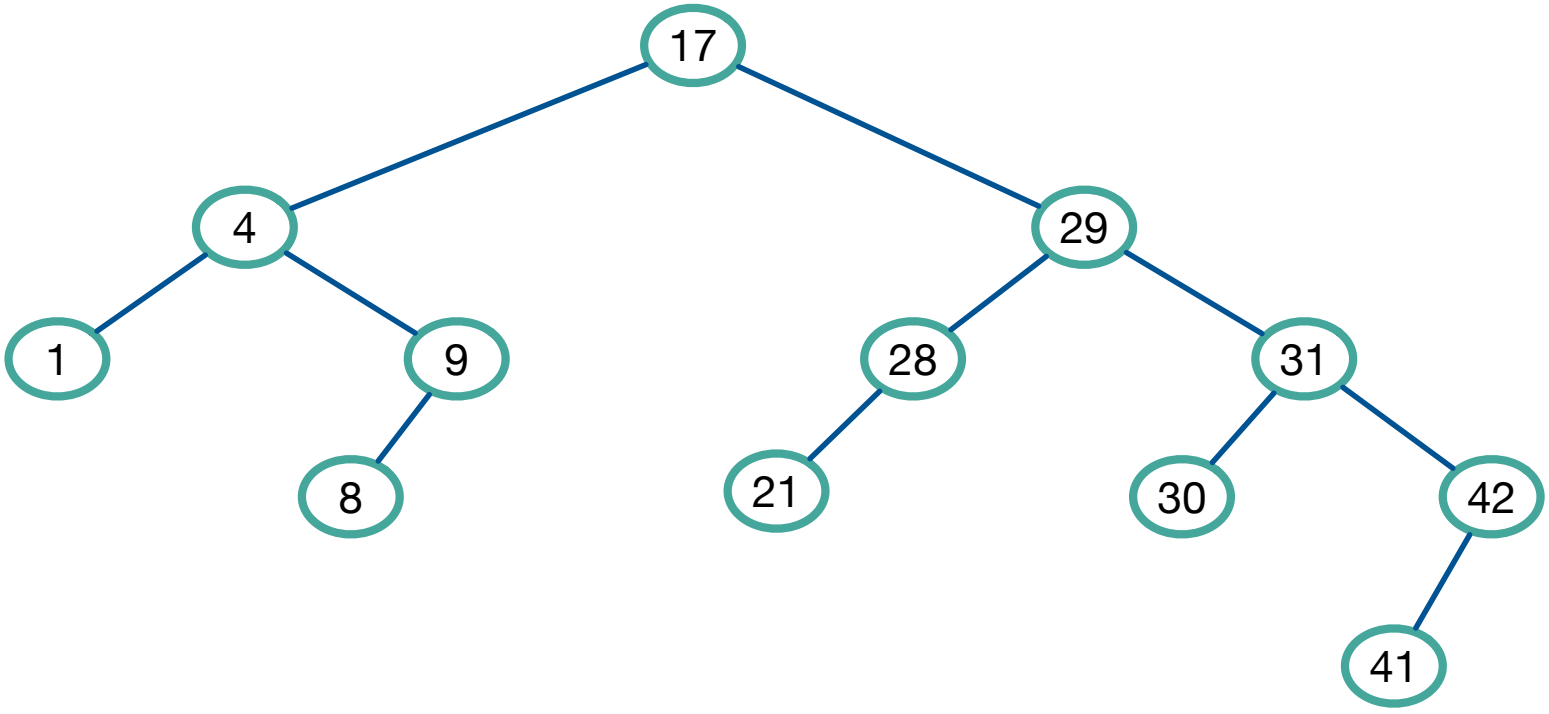
AVL Trees

Is the following an example of an AVL tree? Why or why not?



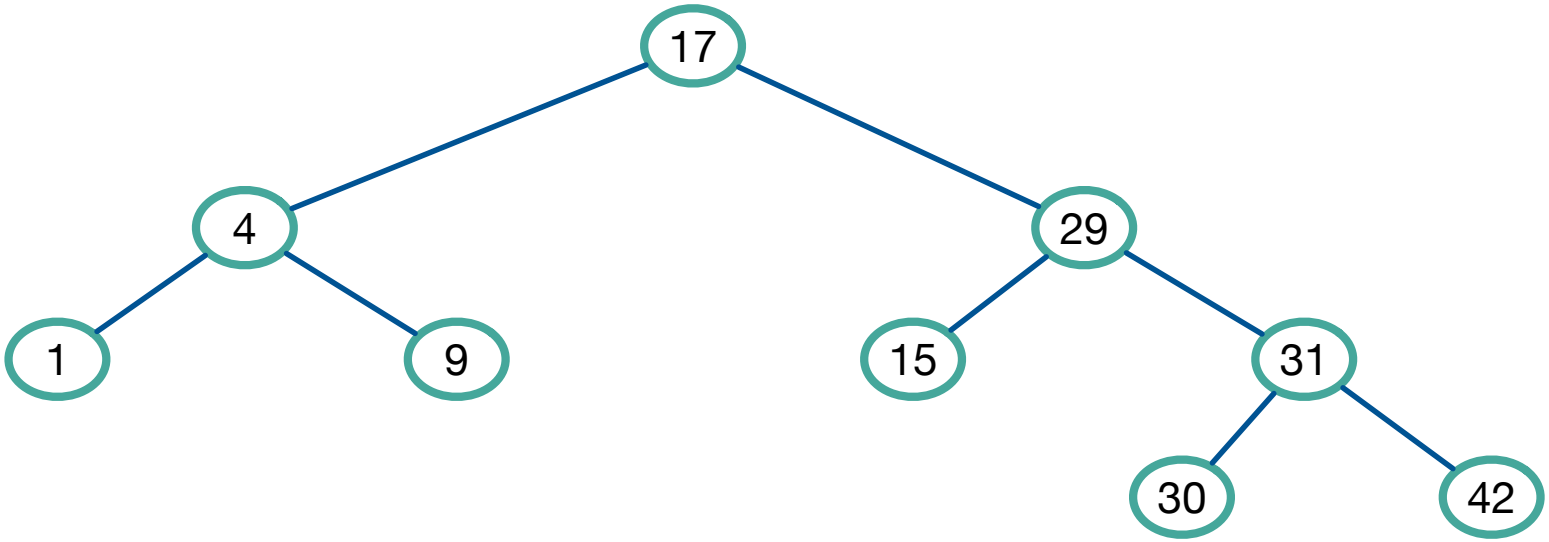
AVL Trees

Is the following an example of an AVL tree? Why or why not?



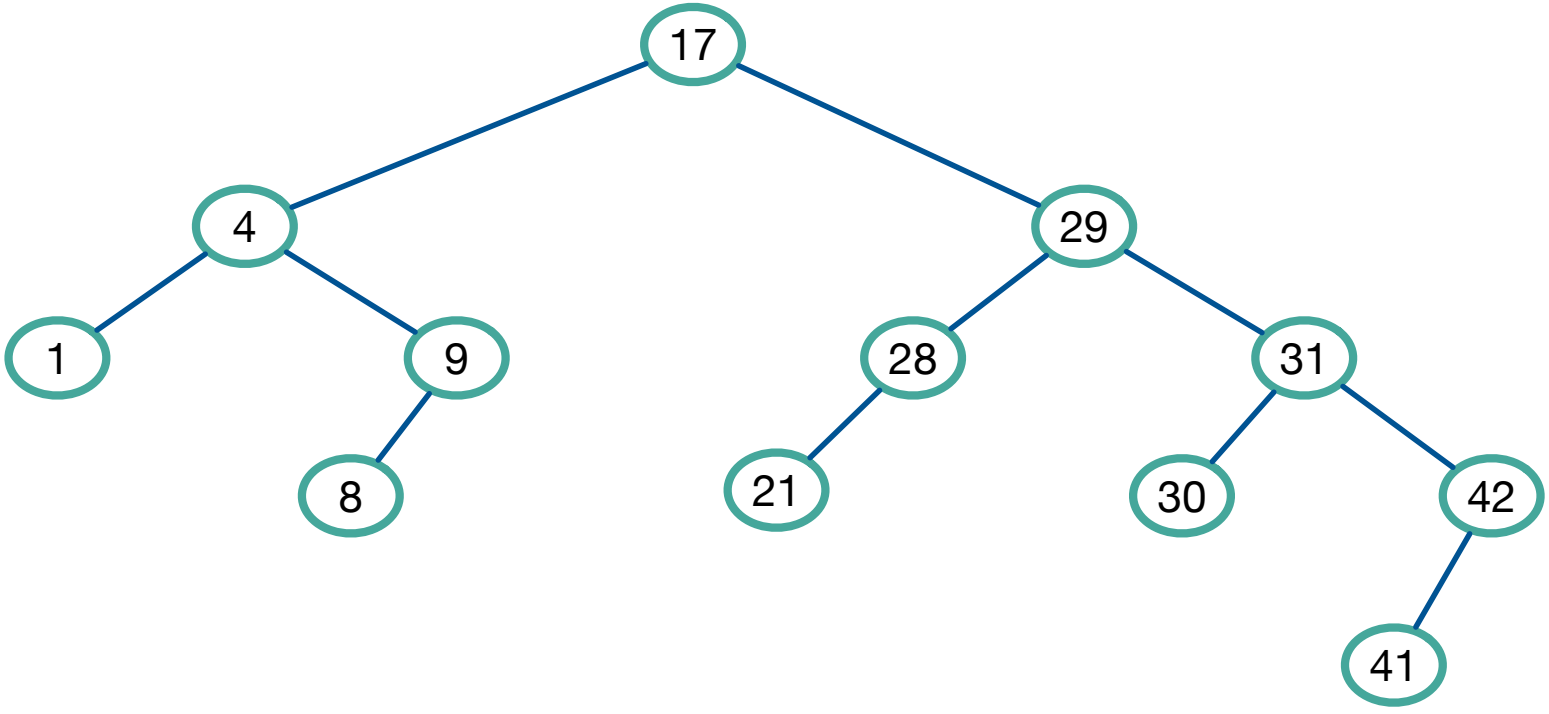
AVL Trees

Is the following an example of an AVL tree? Why or why not?



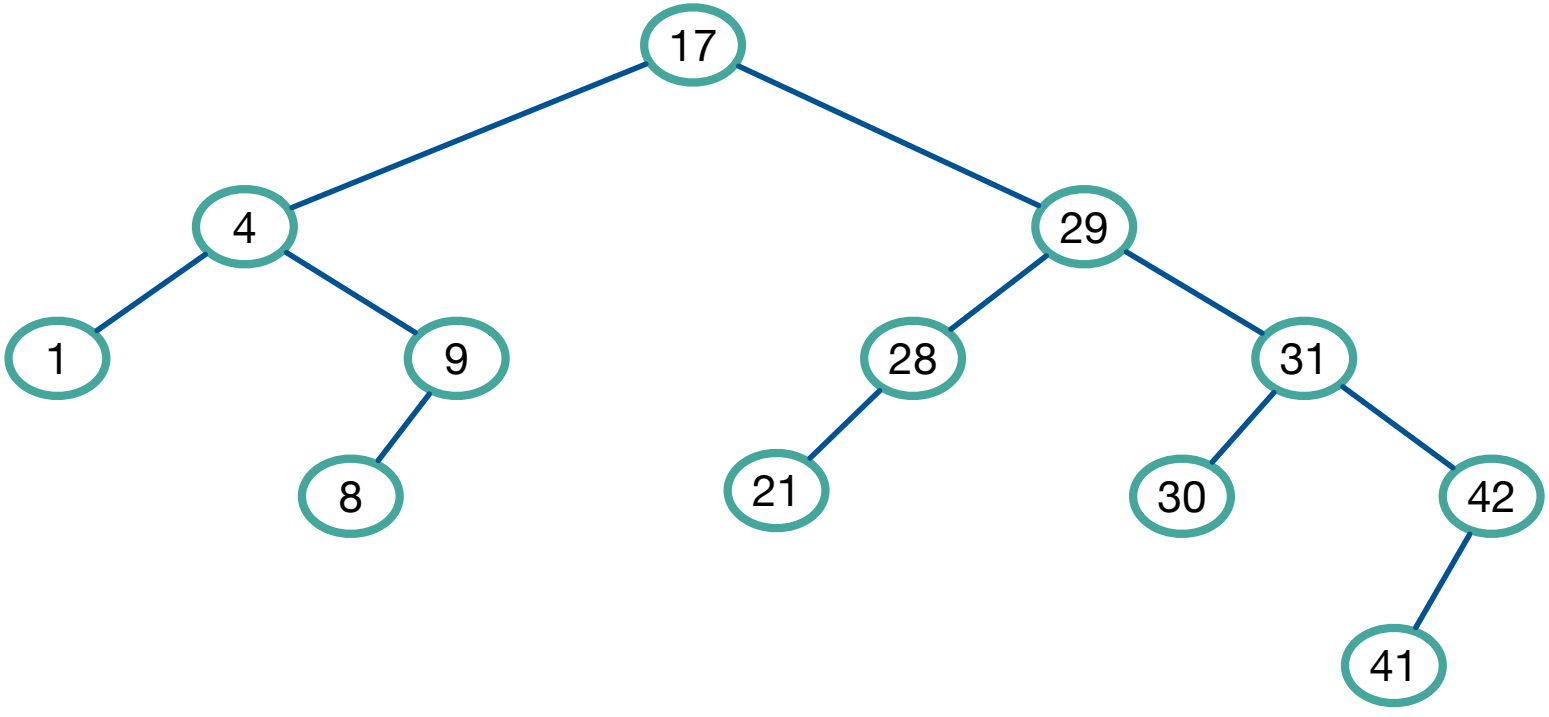
The *Find* Operation

Find is done in the exact same way as it was done in generic binary search trees.



The *Insert* Operation

In what situations would we have to be careful when inserting a vertex?



How would we insert a vertex with key 10? 45? 2? 7?

The *Insert* Operation - Rotations

Suppose, after an insertion, that there is a vertex, x, that is unbalanced.

- Further suppose that x is the deepest unbalanced vertex.
- It must be the case that the height of x's subtrees must differ by two.

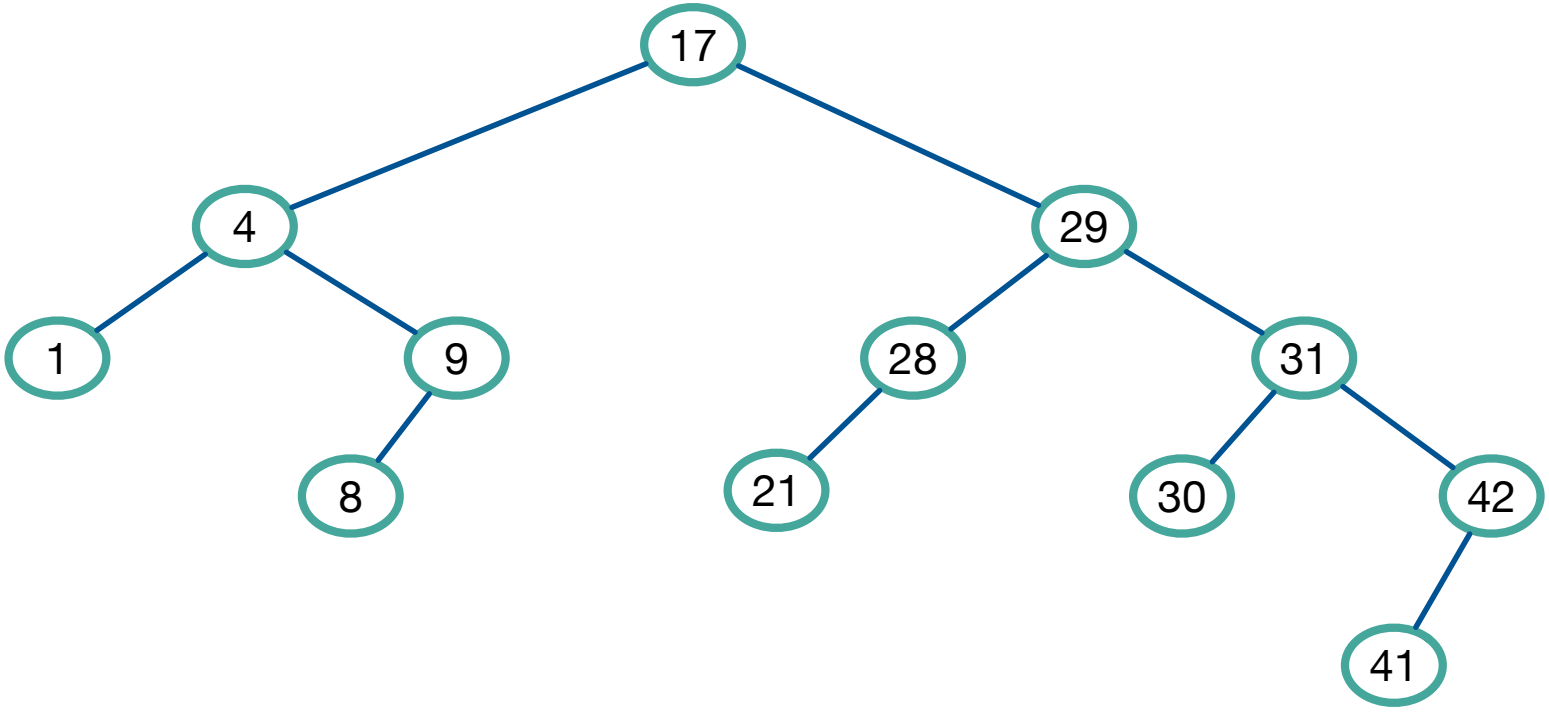
There are four cases for how such an x might come to be:

1. An insertion into the left subtree of the left child of x.
2. An insertion into the right subtree of the left child of x.
3. An insertion into the left subtree of the right child of x.
4. An insertion into the right subtree of the right child of x.

Cases 1 and 4 are symmetric as can be remedied with a *single rotation*.
Cases 2 and 3 are also symmetric and can be remedied with a *double rotation*.

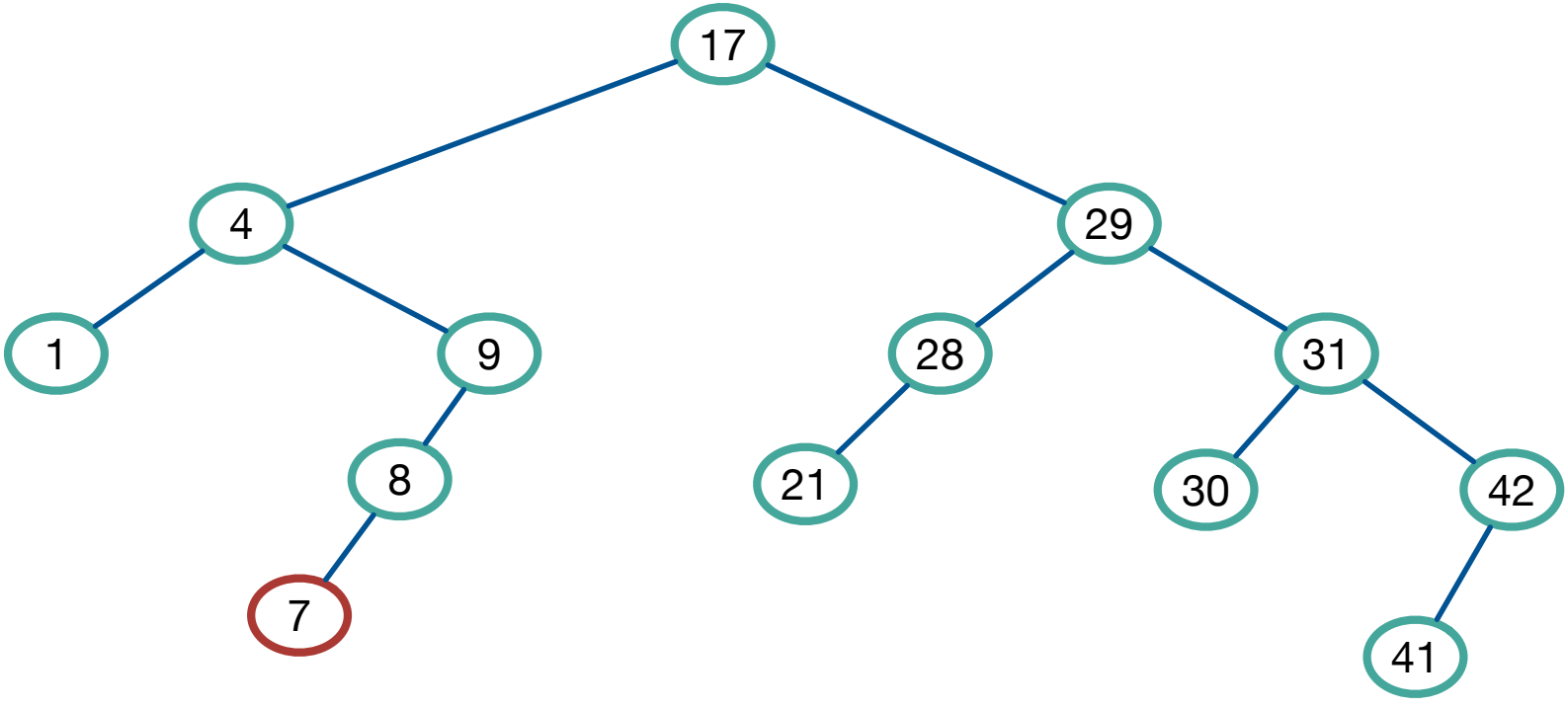
Single Rotation

Suppose we wish to insert a vertex with key 7.



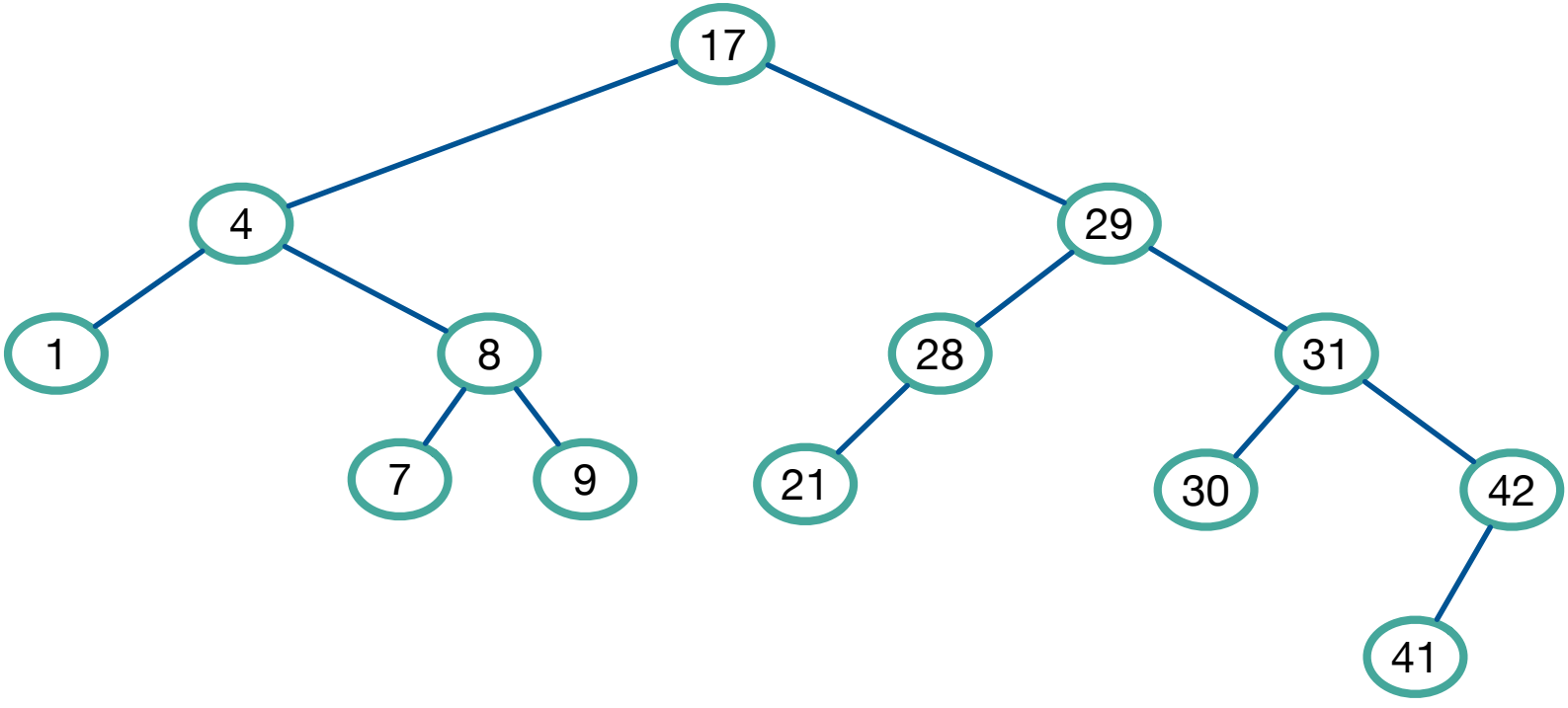
Single Rotation

Suppose we wish to insert a vertex with key 7.



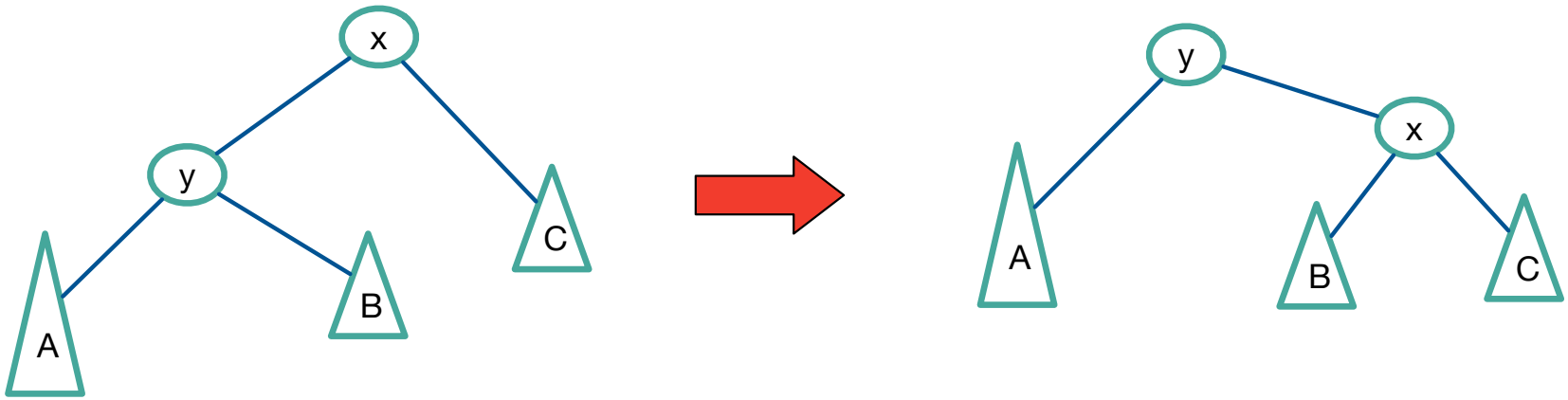
Single Rotation

Suppose we wish to insert a vertex with key 7.



Single Rotation

In general, if inserting into the left subtree of a left child creates an imbalance, do the following:



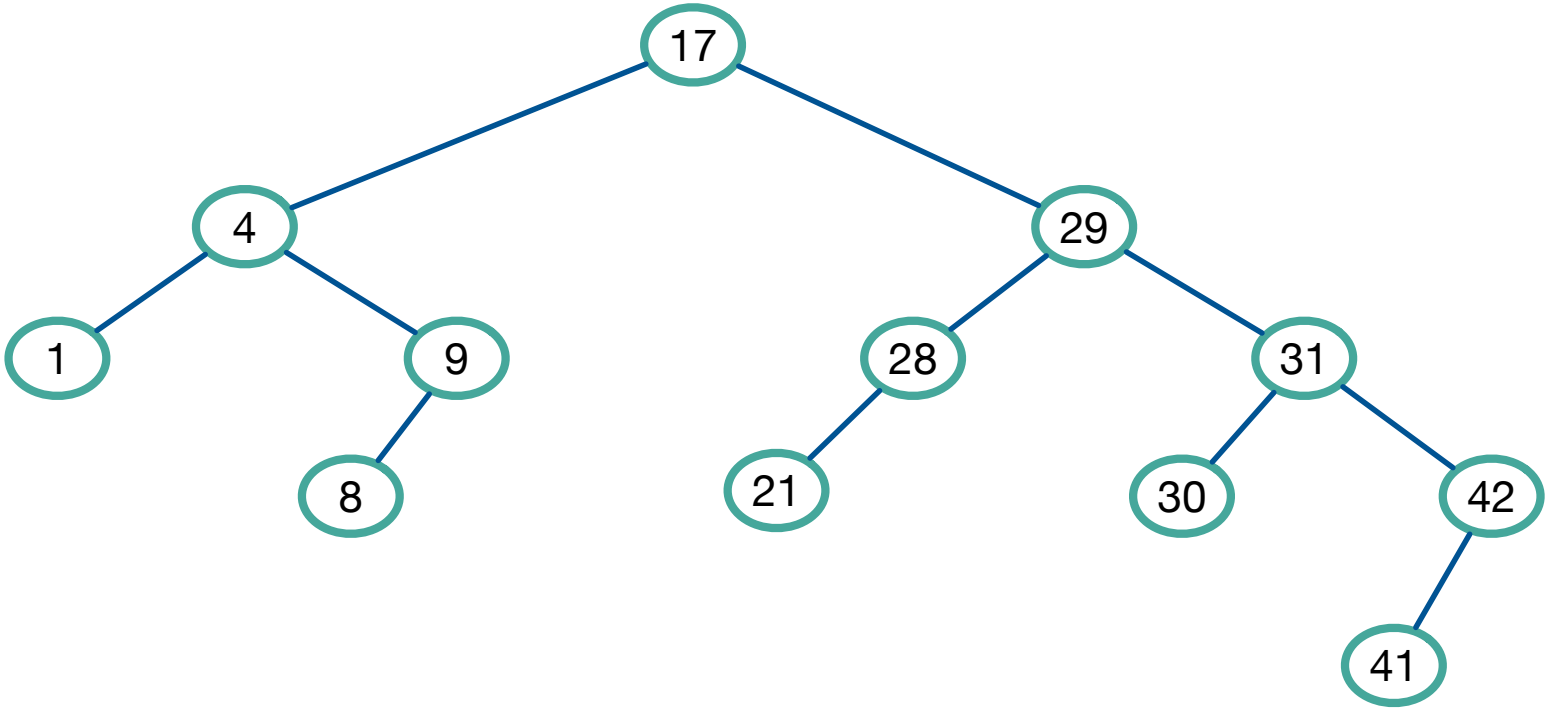
Can you come up with a similar procedure to inserting into the right subtree of the right child?

Single Rotation

Now let's write the pseudocode.

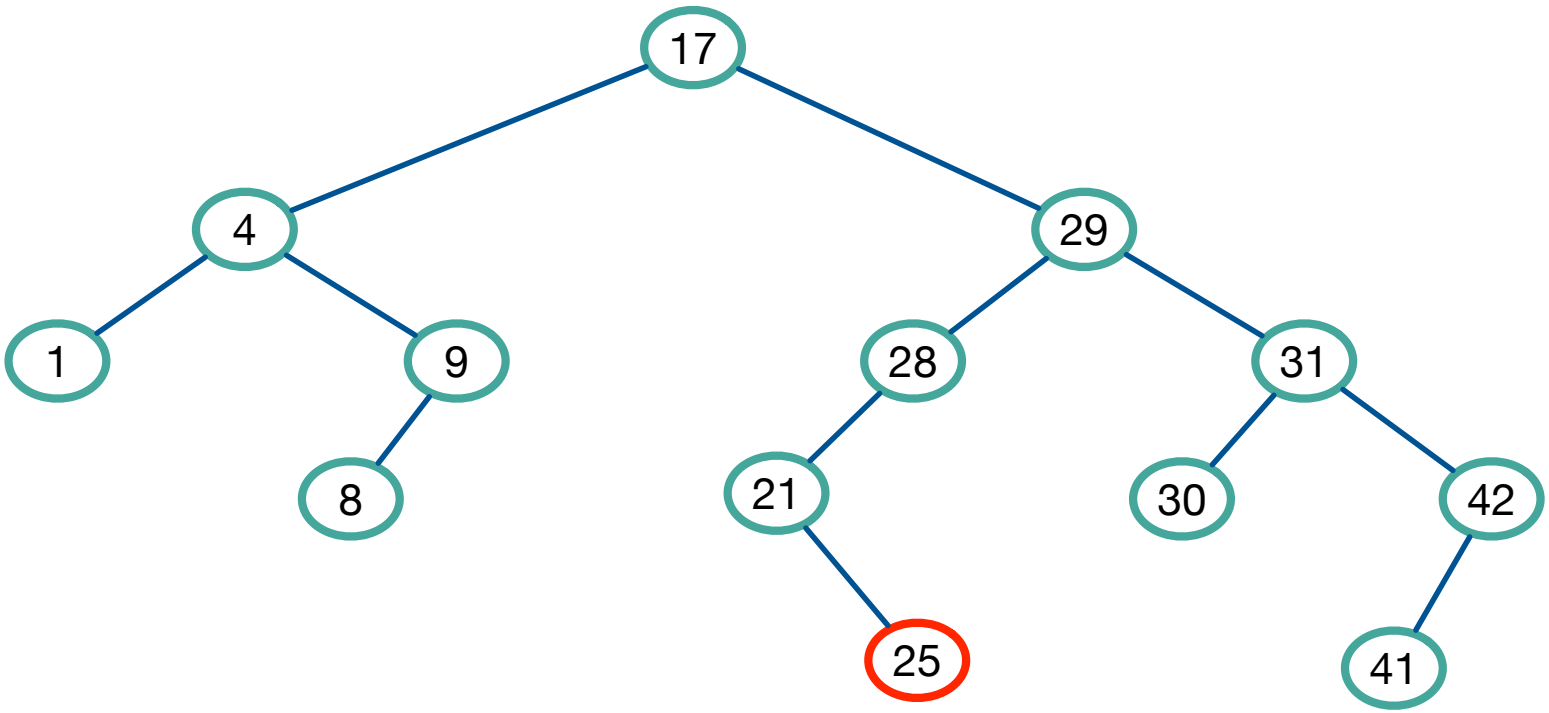
Double Rotation

Suppose we wish to insert a vertex with key 25.

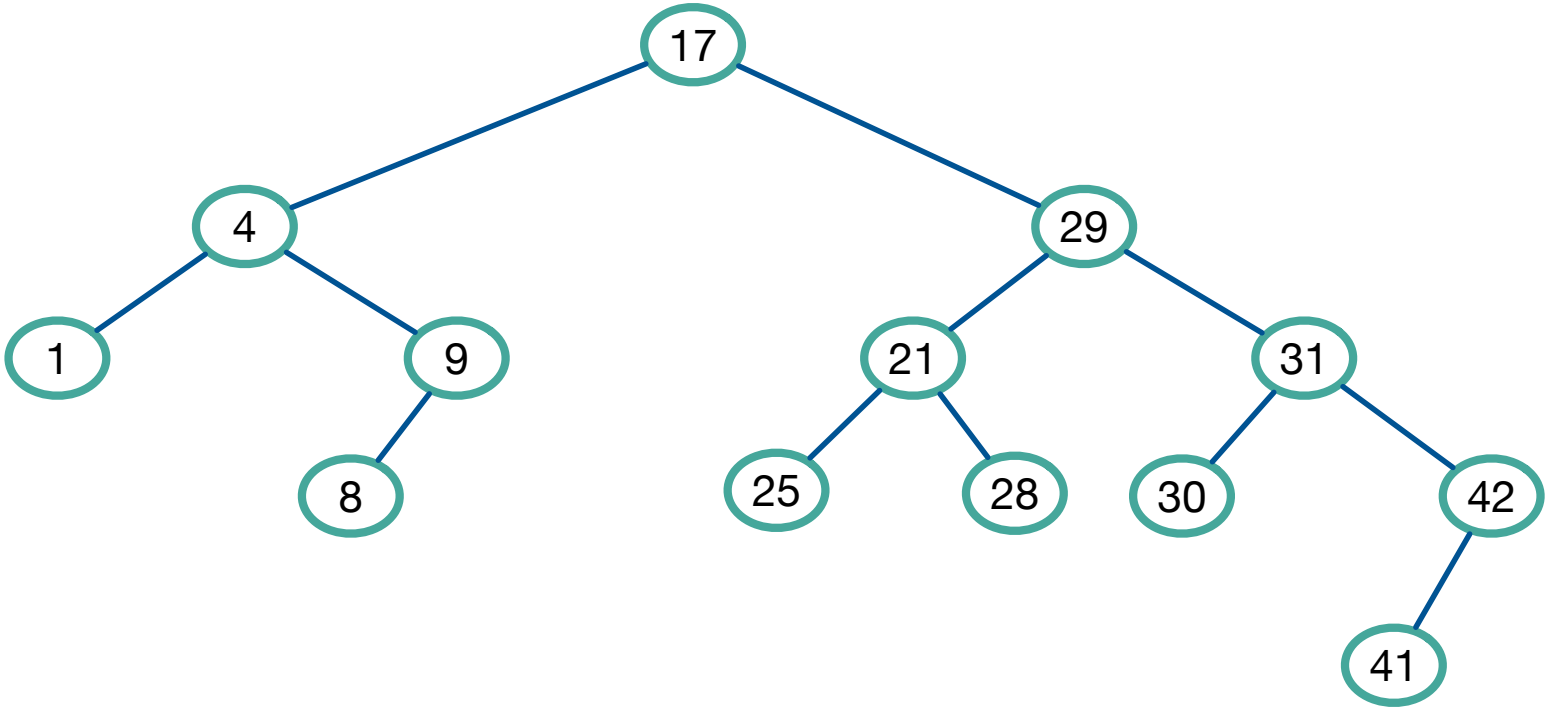


Would the single rotation defined above work?

Double Rotation

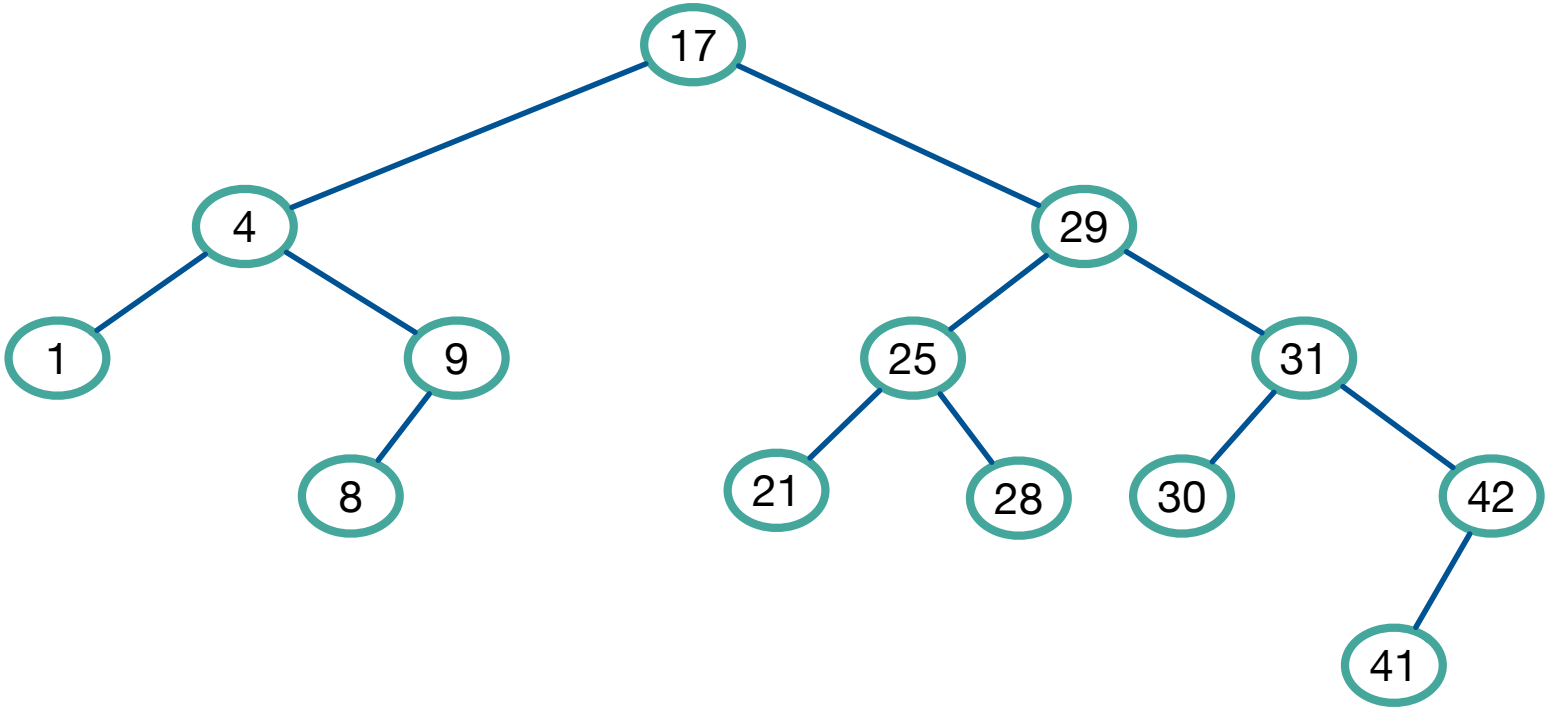


Double Rotation



Is this correct?

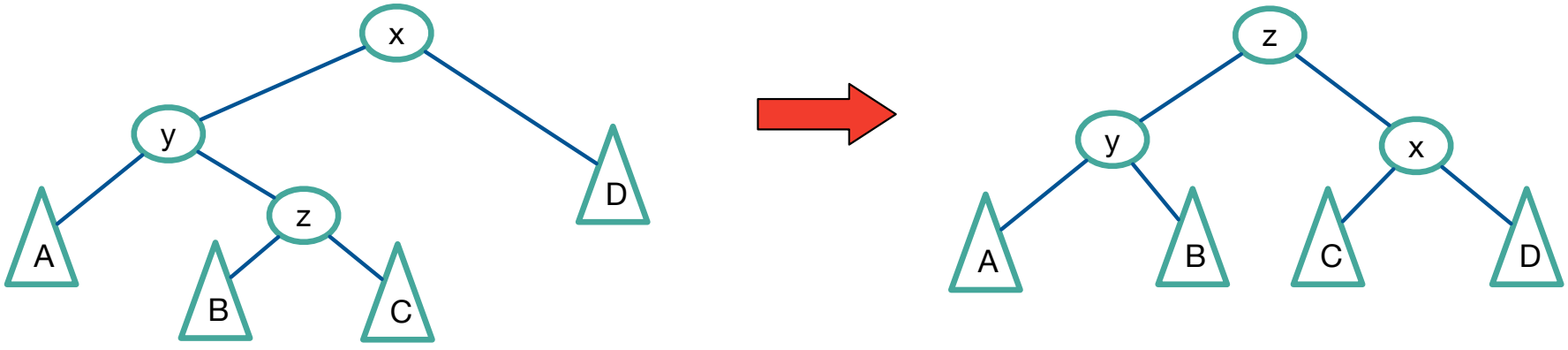
Double Rotation



Is this correct?

Double Rotation

In general, if inserting into the right subtree of a left child creates an imbalance, do the following:



Note that we aren't aware of which subtree of z is deeper, so we picture them both to be the same depth.

Can you come up with a similar procedure to inserting into the left subtree of the right child?

Double Rotation

Now let's write the pseudocode.

Running Times for AVL Trees

- *find:* $O(\text{height of tree}) = O(\log_2 n)$.
- *insert:* $O(\log_2 n)$
- *delete:* $O(\log_2 n)$